Jadavpur University

Department of Computer Science and Engineering

NETWORKS LAB ASSIGNMENT 4

BCSE UG-III

Student : Arjeesh Palai
Roll No. : 002310501086
Group : A3

Date : 03 / 11 / 2025

Assignment 4

Task: Implement Code Division Multiple Access (CDMA) for multiple stations sharing a common channel using *Walsh* codewords. Each sender uses a *unique* Walsh row to spread its bitstream; the receiver reconstructs each sender's data by correlating the composite chips with the corresponding code. Use the same sender–receiver design pattern as earlier labs.

Deliverables: A working sender and receiver, with clear logs showing per-slot composite chips and per-code decoded bits; and a report with figures for the CDMA pipeline and the Walsh table used.

1 Design

Topology. Multiple TCP clients (senders) connect to a single TCP server (receiver) on loopback port 9090. The server represents the shared medium and aggregates chips arriving within a slot.

Codes. We generate an $L \times L$ Walsh matrix H_L ($L = 2^k$) using the Sylvester construction. Sender i uses row $H_L[i]$. A binary bit $b \in \{0, 1\}$ is mapped to BPSK $\{-1, +1\}$ via $b \mapsto (2b - 1)$ and multiplied element-wise with the code to form L chips.

Multiple access superposition. All active senders concurrently transmit their chips; the server sums chipwise to a length-L vector.

Decoding. For code index i with code c_i, the server computes the dot product $d = \sum_j s_j c_{i,j}$. Orthogonality of Walsh rows ensures $\text{sign}(d)$ recovers the transmitted bit for sender i (in the noiseless model).

Timing. A fixed slot length (in ms) buckets arrivals. After each slot, the server prints the composite chips and sends an `ACK 0/1` to each contributing client.

Module responsibilities

- **Sender (client)**: Receive its code assignment from the server (`CODE <idx> <L>`); read bitstrings (from `msg.bits`); for each bit, generate L chips via spreading and stream them; wait for `ACK`.
- **Receiver (server)**: Accept clients, assign successive code indices, accumulate chips per slot, print the composite pattern (`+,-,0`), correlate with each contributing code to decode bits, and `ACK` clients.

2 Implementation (Full Code Snippets)

`common.h` — Walsh prototypes

```
1 #pragma once
2 #include <vector>
```

```
3  #include <string>
4  #include <stdexcept>
5
6  int next_pow2(int x);
7
8  std::vector<std::vector<int>> walsh(int n);
9  std::vector<int> encode_bit(int bit, const std::vector<int> &
       code);
10
11 int decode_bit(const std::vector<int> &chips, const std::
       vector<int> &code);
12 std::string chips_to_wire(const std::vector<int> &chips);
```

Listing 1: common.h

common.cpp — Walsh generator + encode/decode

```
1  #include <vector>
2  #include <string>
3  #include <stdexcept>
4  #include "common.h"
5
6  int next_pow2(int x)
7  {
8    int p = 1;
9    while (p < x)
10     p <<= 1;
11   return p;
12 }
13
14 std::vector<std::vector<int>> walsh(int n)
15 {
16   std::vector<std::vector<int>> H{{1}};
17   while ((int)H.size() < n)
18   {
19     int m = H.size();
20     std::vector<std::vector<int>> T(2 * m, std::vector<int>(2
           * m));
21     for (int i = 0; i < m; i++)
22       for (int j = 0; j < m; j++)
23       {
```

```
24            T[i][j]       = H[i][j];
25            T[i][j + m]   = H[i][j];
26            T[i + m][j]   = H[i][j];
27            T[i + m][j+m] = -H[i][j];
28          }
29        H.swap(T);
30      }
31      return H;
32    }
33
34    std::vector<int> encode_bit(int bit, const std::vector<int> &
        code)
35    {
36      int b = bit ? 1 : -1;
37      std::vector<int> out(code.size());
38      for (size_t i = 0; i < code.size(); ++i)
39        out[i] = b * code[i];
40      return out;
41    }
42
43    int decode_bit(const std::vector<int> &chips, const std::
        vector<int> &code)
44    {
45      long long dp = 0;
46      for (size_t i = 0; i < chips.size(); ++i)
47        dp += 1LL * chips[i] * code[i];
48      return (dp >= 0) ? 1 : 0;
49    }
50
51    std::string chips_to_wire(const std::vector<int> &chips)
52    {
53      std::string s(chips.size(), ' ');
54      for (size_t i = 0; i < chips.size(); ++i)
55        s[i] = (chips[i] >= 0) ? '+' : '-';
56      return s;
57    }
```

Listing 2: common.cpp

receiver.cpp — accept, slot, correlate, ACK

```cpp
#include <bits/stdc++.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "common.h"
using namespace std;

constexpr int PORT = 9090;

int main()
{
  int L, slot_ms;
  cout << "Enter Walsh code length (power of 2): ";
  cin >> L;
  cout << "Enter slot time (ms): ";
  cin >> slot_ms;

  auto H = walsh(L);

  int server_fd = socket(AF_INET, SOCK_STREAM, 0);
  int opt = 1;
  setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt,
     sizeof(opt));
  sockaddr_in addr{};
  addr.sin_family = AF_INET;
  addr.sin_port = htons(PORT);
  addr.sin_addr.s_addr = INADDR_ANY;
  if (bind(server_fd, (sockaddr *)&addr, sizeof(addr)) < 0)
  {
    cerr << "bind failed\n";
    return 1;
  }
  if (listen(server_fd, 64) < 0)
  {
    cerr << "listen failed\n";
    return 1;
  }

  cout << "CDMA Receiver listening on port " << PORT << "
     ...\n";

  mutex mtx, cout_mtx;
  vector<pair<int, int>> clients;     // (socket, code_idx)
```

```
41      unordered_map<int, int> sock2code;  // socket -> code_idx
42      vector<int> slot_sum(L, 0);
43      vector<int> contrib_codes;          // codes that
           contributed this slot
44      atomic<bool> running{true};
45
46      // Accept clients and launch per-connection readers
47      thread acceptor([&]() {
48        int next_code = 0;
49        while (running)
50        {
51          sockaddr_in caddr{}; socklen_t clen = sizeof(caddr);
52          int cs = accept(server_fd, (sockaddr*)&caddr, &clen);
53          if (cs < 0) continue;
54          int code_idx = next_code % L;  // reuse codes if more
               clients than L
55          next_code++;
56
57          {
58            lock_guard<mutex> lk(mtx);
59            clients.push_back({cs, code_idx});
60            sock2code[cs] = code_idx;
61          }
62
63          string hello = string("CODE ") + to_string(code_idx) +
               " " + to_string(L) + "\n";
64          send(cs, hello.c_str(), (int)hello.size(), 0);
65
66          {
67            lock_guard<mutex> lk(cout_mtx);
68            cout << "[ACCEPT] client socket " << cs << " -> code
                 " << code_idx << "\n";
69          }
70
71          // Reader thread per client
72          thread([&, cs, code_idx]() {
73            vector<char> buf(L);
74            while (true)
75            {
76              int got = 0;
77              while (got < L)
78              {
```

```
79              int n = recv(cs, buf.data() + got, L - got, 0);
80              if (n <= 0)
81              {
82                // disconnect
83                lock_guard<mutex> lk(mtx);
84                sock2code.erase(cs);
85                clients.erase(remove_if(clients.begin(),
                     clients.end(),
86                                        [&](auto &p){ return p.
                                             first == cs; }),
87                              clients.end());
88                close(cs);
89                {
90                  lock_guard<mutex> lk2(cout_mtx);
91                  cout << "[DISCONNECT] socket " << cs << " (
                       code " << code_idx << ")\n";
92                }
93                return;
94              }
95              got += n;
96            }
97
98            // Aggregate chips into the shared slot sum
99            {
100             lock_guard<mutex> lk(mtx);
101             for (int i = 0; i < L; i++)
102             {
103               int v = (buf[i] == '+') ? +1 : -1;
104               slot_sum[i] += v;
105             }
106             contrib_codes.push_back(code_idx);
107           }
108         }
109       }).detach();
110     }
111   });
112
113   // Slotter: every slot_ms, print composite and decode per
        contributing code
114   thread slotter([&]() {
115     long long slot_id = 0;
116     while (running)
```

```
117       {
118         this_thread::sleep_for(chrono::milliseconds(slot_ms));
119         vector<int> sum_local;
120         vector<int> contrib_local;
121
122         {
123           lock_guard<mutex> lk(mtx);
124           sum_local = slot_sum;
125           contrib_local.swap(contrib_codes);
126           fill(slot_sum.begin(), slot_sum.end(), 0);
127         }
128         if (contrib_local.empty()) { slot_id++; continue; }
129
130         sort(contrib_local.begin(), contrib_local.end());
131         contrib_local.erase(unique(contrib_local.begin(),
              contrib_local.end()), contrib_local.end());
132
133         // Print composite chips
134         {
135           lock_guard<mutex> lk(cout_mtx);
136           cout << "[SLOT " << slot_id << "] composite chips: ";
137           for (int i = 0; i < L; i++)
138           {
139             int v = sum_local[i];
140             if (v > 0)      cout << '+';
141             else if (v < 0) cout << '-';
142             else            cout << '0';
143           }
144           cout << "\n";
145         }
146
147         // Decode per contributing code and ACK
148         for (int code_idx : contrib_local)
149         {
150           int bit = decode_bit(sum_local, H[code_idx]);
151           int cs = -1;
152           {
153             lock_guard<mutex> lk(mtx);
154             for (auto &p : clients)
155               if (p.second == code_idx) { cs = p.first; break;
                    }
156           }
```

```
157           if (cs != -1)
158           {
159             string ack = string("ACK ") + char('0' + bit);
160             send(cs, ack.c_str(), (int)ack.size(), 0);
161           }
162           {
163             lock_guard<mutex> lk(cout_mtx);
164             cout << "         decode for code " << code_idx <<
                  " => bit " << bit << "\n";
165           }
166         }
167         slot_id++;
168       }
169     });
170
171     acceptor.join();
172     slotter.join();
173     close(server_fd);
174     return 0;
175   }
```

Listing 3: receiver.cpp

sender.cpp — connect, spread, send, await ACK

```
1   #include <bits/stdc++.h>
2   #include <arpa/inet.h>
3   #include <unistd.h>
4   #include "common.h"
5   using namespace std;
6
7   constexpr int PORT = 9090;
8   string SERVER_IP = "127.0.0.1";
9
10  int main()
11  {
12    int n_clients, frames;
13    cout << "Enter number of clients: ";
14    cin >> n_clients;
15    cout << "Frames per client: ";
16    cin >> frames;
```

```
17
18      ifstream fin("msg.bits");
19      vector<string> msgs;
20      string line;
21      while (getline(fin, line))
22        if (!line.empty()) msgs.push_back(line);
23      fin.close();
24      if (msgs.empty())
25      {
26        cerr << "msg.bits empty!\n";
27        return 1;
28      }
29
30      mutex cout_mtx;
31      atomic<long long> bits_sent{0}, bits_acked{0};
32
33      auto worker = [&](int id)
34      {
35        int sock = socket(AF_INET, SOCK_STREAM, 0);
36        sockaddr_in serv{};
37        serv.sin_family = AF_INET;
38        serv.sin_port = htons(PORT);
39        inet_pton(AF_INET, SERVER_IP.c_str(), &serv.sin_addr);
40        if (connect(sock, (sockaddr *)&serv, sizeof(serv)) < 0)
41        {
42          lock_guard<mutex> lk(cout_mtx);
43          cerr << "[C" << id << "] connect failed\n";
44          return;
45        }
46
47        // Read CODE line
48        string hello; hello.reserve(64);
49        char tmp[64];
50        while (true)
51        {
52          int n = recv(sock, tmp, sizeof(tmp), 0);
53          if (n <= 0)
54          {
55            lock_guard<mutex> lk(cout_mtx);
56            cerr << "[C" << id << "] no CODE line\n";
57            close(sock);
58            return;
```

```
59          }
60          hello.append(tmp, tmp + n);
61          if (hello.find('\n') != string::npos) break;
62        }
63        int code_idx = -1, L = -1;
64        sscanf(hello.c_str(), "CODE %d %d", &code_idx, &L);
65        auto H = walsh(L);
66        const auto &code = H[code_idx];
67
68        {
69          lock_guard<mutex> lk(cout_mtx);
70          cout << "[C" << id << "] assigned code " << code_idx <<
                " (L=" << L << ")\n";
71        }
72
73        for (int f = 0; f < frames; ++f)
74        {
75          const string &bits = msgs[f % msgs.size()];
76          for (char bch : bits)
77          {
78            int bit = (bch == '1') ? 1 : 0;
79            auto chips = encode_bit(bit, code);
80            string wire = chips_to_wire(chips);
81
82            {
83              lock_guard<mutex> lk(cout_mtx);
84              cout << "[C" << id << "] send bit=" << bit << "
                    chips=" << wire << "\n";
85            }
86
87            // send all chips for this bit
88            const char *p = wire.data();
89            int left = (int)wire.size();
90            while (left > 0)
91            {
92              int n = send(sock, p, left, 0);
93              if (n <= 0)
94              {
95                close(sock);
96                return;
97              }
98              p += n;
```

```
99              left -= n;
100           }
101           bits_sent++;
102
103           // wait for ACK
104           char abuf[16];
105           int n = recv(sock, abuf, sizeof(abuf), 0);
106           if (n > 0)
107           {
108             string s(abuf, abuf + n);
109             int rec_bit = -1;
110             if (s.size() >= 5 && s.rfind("ACK ", 0) == 0)
111               rec_bit = (s[4] == '1') ? 1 : 0;
112             bits_acked++;
113             {
114               lock_guard<mutex> lk(cout_mtx);
115               cout << "[C" << id << "] ACK recv decoded_bit="
116                    << (rec_bit == -1 ? '?' : ('0' + rec_bit))
                        << "\n";
117             }
118           }
119         }
120       }
121       close(sock);
122     };
123
124     vector<thread> th;
125     for (int i = 0; i < n_clients; i++)
126       th.emplace_back(worker, i);
127     for (auto &t : th)
128       t.join();
129
130     cout << "\n--------------Sender Result--------------\n";
131     cout << "Bits that were sent:  " << bits_sent.load() << "\n
          ";
132     cout << "Bits in which ACK received: " << bits_acked.load()
          << "\n";
133     return 0;
134   }
```

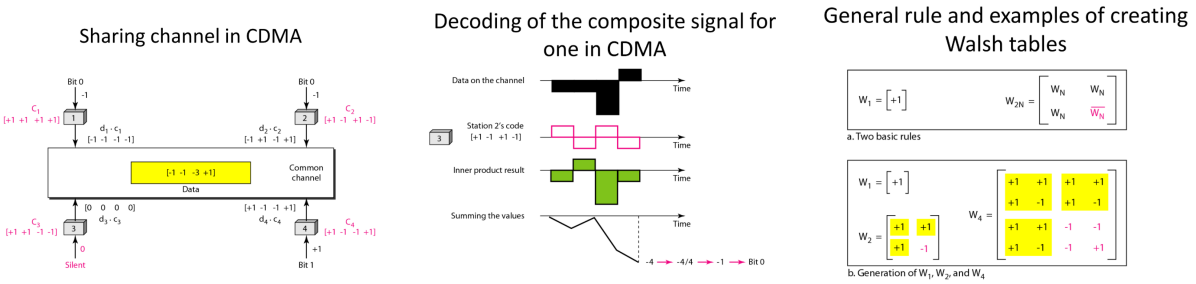
Listing 4: sender.cpp

3 Build & Run (sample)

Linux / macOS (C++17)

```
# Terminal 1: build + run receiver (shared medium)

g++ -std=c++17 -O2 -pthread receiver.cpp common.cpp -o receiver
./receiver

# Enter, e.g.: L=8, slot_ms=60

# Terminal 2: build + run sender(s)

g++ -std=c++17 -O2 -pthread sender.cpp common.cpp -o sender
./sender

# Enter, e.g.: n_clients=4, frames=3

# Ensure a file 'msg.bits' (one bitstring per line) is present.
```

4 Diagrams of CDMA and Walsh Table



CDMA sender/receiver overview. CDMA decoding. Walsh table (example with $L = 4$).

Figure 1: CDMA and Walsh table diagrams.

5 Appendix: Key Parameters (as used)

- Walsh length L: power of two (e.g., 4, 8, 16). Slot time: user input in ms.
- Sender inputs: number of clients, frames per client, and `msg.bits` file with the payload bitstrings.

- Receiver logs: per-slot composite chips and per-code decoded bit (`ACK 0/1`).